Amendments to the Specification:

Please replace the title with the following replacement title:

--METHOD FOR INCREASING CONFIGURATION RUNTIME OF TIME-SLICED CONFIGURATIONS--.

Please replace the paragraph beginning at page 5, line 4 with the following replacement paragraph:

--Figures 1a-1fII include ~~Figure 1 includes~~ diagrams illustrating passing of data between a data processing logic cell field and memory, according to exemplary embodiments of the present invention.--.

Please replace the paragraph beginning at page 5, line 8 with the following replacement paragraph:

--Figures 3a-3d include ~~Figure 3 includes~~ diagrams that illustrate different arrangements of FPGAs and ALUs and/or EALUs of a logic cell field, according to exemplary embodiments of the present invention.--.

Please replace the paragraph beginning at page 5, line 15 with the following replacement paragraph:

--Figures 6a to 6b [[6c]] are diagrams and a table that illustrate a task switch, a thread switch, and/or a hyperthread switch, according to exemplary embodiments of the present invention.--.

Please insert the following new paragraph at page 5, line 17, immediately preceding the "Detailed Description of the Invention" section:

--Figure 7 is a flowchart illustrating a method for a configuration to increase its maximum allowed runtime, according to an example embodiment of the present invention.--.

Please replace the paragraph beginning at page 5, line 18 with the following replacement paragraph:

-- In an example embodiment of the present invention, data may be supplied to the data processing logic cell field in response to execution of a load configuration by the data processing logic cell field, and/or data from this data processing logic cell field may be written back (STORED) by processing a STORE configuration accordingly. These load

configurations and/or memory configurations may be designed in such a way that addresses of memory locations to be accessed directly or indirectly by loading and/or storage are generated directly or indirectly within the data processing logic cell field. Through this configuration of address generators within a configuration, a plurality of data may be loadable into the data processing logic cell field, where it may be stored in internal memories (iRAM), if necessary, and/or in internal cells such as EALUs having registers and/or internal memory arrangements. The load configuration and/or memory configuration may thus allow loading of data by blocks, almost like data streaming, in particular being comparatively rapid in comparison with individual access, and such a load configuration may be executable before one or more configurations that process data by actually analyzing and/or modifying it, with which configuration(s) the previously loaded data is processed. Data loading and/or writing may typically take place in small areas of large logic cell fields, while other subareas may be involved in other tasks. Reference is made to Figures 1a-1fII ~~Figure 1~~ for these and other particulars of the present invention. In the ping-pong-like data processing described in other published documents by the present applicant in which memory cells are provided on both sides of the data processing field, one memory side may be preloaded with new data by a LOAD configuration in an array part, while data from the opposite memory side having a STORE configuration may be written back in another array part; in a first processing step. Data from the memory on one side may stream through the data processing field to the memory on the other side. Intermediate results obtained in the first stream through the field may be stored in the second memory, the field may be reconfigured, if necessary, and the interim results may then stream back for further processing, etc. This simultaneous LOAD/STORE procedure is also possible without any spatial separation of memory areas.--.

Please insert the following paragraphs immediately preceding the paragraph beginning at page 6, line 15:

--For example, Figure 1a provides an overview of the basic data operation model. Four listed configurations, config 1,2,3,n, are shown to be loaded via a Configuration Manager (CT) into a reconfigurable array, which includes at least two banks of iRAMs, one at each side. The listed configurations config 1,2,3,n are shown in detail in the Figures 1bI-1fII. Address generators (labeled as In/Out Addressgen or IOAGs) transfer internal data to the array or iRAMs from, or from the array or iRAMs to, external elements, such as external memory.

Figure 1bI shows a configuration loading data from external memory into one or more internal iRAM-Bank(s) via the IOAG. The IOAG is shown to generate addresses (A) for the external memory, which, in turn, in shown to return the according data (D) to the IOAG. The IOAG forwards the data, e.g., through the reconfigurable array, to the iRAM-Bank(s). The configuration executed on the reconfigurable array may comprise further address generation, which may be fed to the IOAG, and/or provide addressing for one or multiple iRAM-Bank(s). Also, according control for loading the data is shown to be provided.

Figure 1bII shows an alternative implementation, where a Cache Controller is used, e.g., in place of the IOAG. The Load/Store configuration, according to Figure 1bI provides addresses to the Cache controller. A cache hierarchy, shown to include Level 1-n caches, connects the reconfigurable array with the external memory.

Figure 1bIII shows the same, but with a separated IOAG between the reconfigurable array and the Cache.

Figure 1c shows a first data processing step, in which data to be processed is read from a source iRAM (iRAM-Bank1), which may be loaded as described above. The data is processed by execution within the configurable array, while additional input data may be received through the IOAG from an external source. Result data is written to a target iRAM (iRAM-Bank2) and/or may be sent out through the IOAG. The configurations of Figures 1c-1d are described as analogous to the game of "ping pong." The configuration of Figure 1c is therefore referenced as "ping" as data is transferred in a first direction from left to right.

The subsequent configuration shown in Fig 1d is called "pong" as the next processing step reads from iRAM-Bank2 the result data previously produced in the "ping" step, using the result data as input data. The input data is processed, again possibly together with additional data from the IOAG, and the results are written into iRAM-Bank1, while (again) some data might be sent out to external devices via the IOAG.

As shown in Figure 1eI, operand data might be read from one iRAM-Bank and be written back into the same iRAM-Bank.

As shown in Figure 1e2, for example, two configurations Config1 and Config2 can operate in parallel and access one or multiple iRAM-Banks, e.g., in parallel.

Figure 1fII shows the function of a Store configuration, reading data from an iRAM-Bank and writing the data to the external memory. The according configuration is shown in Figure 1fI, essentially including the same elements and functions as Figure 1bI in the reverse direction, i.e., such that data is read from the iRAM-Bank(s) and sent via the IOAG to the external memory. Further address calculation might be provided by the reconfigurable array

for the iRAM-Bank(s) and/or the IOAG for addressing the external memory. Store control might control the data transfer from the iRAM-Bank(s) to or through the IOAG.--.

Please replace the paragraph beginning at page 10, line 10 with the following replacement paragraph:

--In addition to blockwise and/or streaming and/or random reading and/or writing access, in particular in read-modify-write mode (RMW) mode to cache areas and/or the LOAD/STORE unit and/or the connection (known per se in the related art) to the register of the sequential CPU, there may also be a connection to an external bulk memory such as a RAM, a hard drive and/or another data exchange port such as an antenna, etc. A separate port may be provided for this access to cache arrangements and/or LOAD/STORE units and/or memory arrangements different from register units. Suitable drivers, buffers, signal processors for level adjusting and so forth may be provided, e.g., LS74244, LS74245. The logic cells of the field may include ALUs and/or EALUs, in particular but not exclusively for processing a data stream flowing in or into the data processing logic cell field, and typically short fine-granularly configurable FPGA type circuits may be provided upstream from them at the inlet and/or outlet ends, in particular at both the inlet and outlet ends, and/or may be integrated into the PAE-ALU to cut bit blocks out of a continuous data stream, for example, as is necessary for MPEG4 decoding. This may be advantageous when a data stream is to enter the cell and is to be subjected there to a type of preprocessing without blocking larger PAEs units of this type. This may also be of particular advantage when the ALU is designed as a SIMD arithmetic unit, in which case a very long data input word having a data length of 32 bits, for example, may then be split up via the upstream FPGA-type strips into a plurality of parallel data words having a length of 4 bits, for example, which may then be processed in parallel in the SIMD arithmetic units, which is capable of significantly increasing the overall performance of the system, if corresponding applications are needed. FPGA-type upstream and/or downstream structures were discussed above. However, FPGA-type does not necessarily refer to 1-bit granular arrangements. It is possible in particular to provide, instead of these hyperfine granular structures, only fine granular structures having a width of 4 bits, for example. In other words, FPGA-type input and/or output structures upstream and/or downstream from an ALU unit designed as a SIMD arithmetic unit in particular may be configurable, for example, so that 4-bit data words are always supplied and/or processed. It may be possible to provide cascading here so that, for example, the incoming 32-bit-long data words stream into four separate and/or separating 8-bit FPGA-type structures positioned side

by side, a second strip having eight 4-bit-wide FPGA-type structures is downstream from these four 8-bit-wide FPGA-type structures and then, if necessary, after another such strip, if necessary for the particular purpose, sixteen parallel 2-bit wide FPGA-type structures are also provided side by side, for example. If this is the case, a substantial reduction in configuration complexity may be achieved in comparison with strictly hyperfine granular FPGA-type structures. This may also result in the configuration memory of the FPGA-type structure possibly turning out to be much smaller, thus permitting a savings in terms of chip area. FPGA-type strip structures, as also shown in conjunction with ~~Figure 3~~ Figures 3a-3d, in particular situated in the PAE, may permit implementation of pseudo-random noise generators in a particularly simple manner. In an example embodiment of the present invention, if individual output bits obtained stepwise always from a single FPGA cell are written back to the FPGA cell, a pseudo-random noise may also be generated creatively using a single cell (see Figure 5).--.

Please replace the paragraph beginning at page 11, line 29 with the following replacement paragraph:

--The cache need not necessarily be divided into slices, and if this is the case, a separate thread need not necessarily be assigned to each slice. Further, there may be cases in which not all cache areas are being used simultaneously or temporarily at a given point in time. Instead, it is to be expected that in typical data processing applications such as those occurring with handheld mobile telephone (cell phones), laptops, cameras and so forth, there are frequently times during which the entire cache is not needed. Therefore, in an example embodiment of the present invention, individual cache areas may be separable from the power supply so that their power consumption drops significantly, in particular to zero or almost zero. In a slice-wise cache design, this may occur by shutting down the cache in slices via suitable power disconnection arrangements ~~(see Figure 2, for example)~~. The disconnection may be accomplished either by cycling down, clock disconnection, or power disconnection. For example, Figure 2 shows cache slices and separately controllable connections to power and clock signals via multiplexers and switches, respectively. In particular, access recognition may be assigned to an individual cache slice or the like, this access recognition being designed to recognize whether a particular cache area, *i.e.*, a particular cache slice, has a thread, hyperthread, or task assigned to it at the moment, by which it is being used. If the access recognition then ascertains that this is not the case, typically disconnection from the clock and/or even from the power may then be possible. On

reconnecting the power after a disconnection, immediate response of the cache area may be possible again, *i.e.*, no significant delay need be expected due to turning the power supply on and off if implemented in hardware using conventional suitable semiconductor technologies. This is appropriate in many applications independently of the use with logic cell fields.--.

Please replace the paragraph beginning at page 15, line 23 with the following replacement paragraph:

--Reference was made above to data processing logic cell fields which are runtime reconfigurable in particular. The fact that a configuration management unit (CT and/or CM) may be provided for these systems was discussed. Management of configurations per se is known from the various patents and applications by the present applicant, to which reference has been made for disclosure purposes, as well as the applicant's other publications. Such units and their mechanism of operation via which configurations not yet currently needed are preloadable, in particular independently of connections to sequential CPUs, etc., may also be highly usable for inducing a task switch, a thread switch, and/or a hyperthread switch in multitasking operation, in hyperthreading, and/or in multithreading (see Figures 6a through 6b [[6c]], for example). That, during the runtime of a thread or task, configurations for different tasks, *i.e.*, threads and/or hyperthreads, may also be loaded into the configuration memory in the case of a single cell or a group of cells of the data processing logic cell field, *i.e.*, a PAE of a PAE field (PA), for example, may be used to do so. That is, in the case of a blockade of a task or thread, e.g., when it is necessary to wait for data because the data is not yet available, whether because it has not yet been generated or received by another unit, e.g., because of latencies, or because a resource is currently still being blocked by another access, configurations for another task or thread may be preloadable and/or preloaded and it is possible to switch to them without the time overhead of having to wait for a configuration switch in the case of a shadow-loaded configuration in particular. In principle, it is possible to use this technique even when the most probable continuation is predicted within a task and a prediction is not correct (prediction miss), but this type of operation is preferred in prediction-free operation. In the case of use with a purely sequential CPU and/or multiple purely sequential CPUs, in particular exclusively with such CPUs, multithreading management hardware may thus be implemented by adding a configuration manager. Reference is made in this regard in particular to PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120) and PACT17 (DE 100 28 397.7, WO 02/13000). It may be regarded as sufficient, in particular if hyperthreading management is desired for a CPU and/or a few

sequential CPUs, to omit certain partial circuits like the FILMO as described in the patents and applications to which reference has been made specifically. In particular, this also describes the use of the configuration manager described there with and/or without FILMO for hyperthreading management for one or more purely sequentially operating CPUs with or without connection to an XPP or another data processing logic cell field. A plurality of CPUs may be implemented using the known techniques, as are known in particular from PACT31 (DE 102 12 621.6-53, PCT/EP 02/10572) and PACT34 (DE 102 41 812.8, PCT/EP 03/09957) in which one or more sequential CPUs are provided within an array, utilizing one or more memory areas in the data processing logic cell field in particular for construction of the sequential CPU, in particular as an instruction register and/or data register. It should also be pointed out here that previous patent applications such as PACT02 (DE 196 51 075.9-53, WO 98/26356), PACT04 (DE 196 54 846.2-53, WO 98/29952), and PACT08 (DE 197 04 728.9, WO 98/35299) have already disclosed how sequencers having ring and/or random access memories may be constructed.--.

Please replace the paragraph beginning at page 19, line 19 with the following replacement paragraph:

--In systems which must respond to interrupts more quickly, in one embodiment of the present invention, a single resource, *i.e.*, for example, a separate XPP unit and/or parts of an XPP field, may be reserved for such processing. If an interrupt which must be processed quickly then occurs, it is possible to either process a configuration preloaded for particularly critical interrupts in advance or to begin immediately loading an interrupt processing configuration into the reserved resource. A choice of the particular configuration required for the corresponding interrupt is possible through appropriate triggering, wave processing, etc. Thus, with reference to Figure 7, at step 700, processing may be begin according to a configuration, in response to which a counter may be enabled to begin counting at step 701. During the processing, the configuration may, at step 705 determine whether to retrigger the counter to increase its maximum allowed time. If it is determined that the configuration should end and the CT should perform a reconfiguration, the counter may continue without being reset until maximum runtime of the current configuration is reached at step 703. Responsive to reaching the maximum runtime, the CT may load a new configuration at step 704. If it is determined at step 705 that the configuration should be continued, the configuration may retrigger the counter at step 706. If an interrupt is detected at step 707, the trigger of the configuration may be suppressed, so that maximum runtime is reached at step 703. If an interrupt is not detected, the configuration's trigger may reset the counter at step 701, and the process may be repeated.--.